título

Procesamiento masivo de datos mediante Cassandra y Spark

Máster en Sistemas Informáticos Avanzados
Septiembre de 2016

Autor:

Xabier Zabala Barandiaran

Supervisores:

German Rigau i Claramunt
UPV/EHU
Iñigo Etxabe y Beñat Aranburu
Datik Información Inteligente S.L.

Agradecimientos

En primer lugar, quisiera expresar mi gratitud hacia las personas que han posibilitado la concepción y el desarrollo de este proyecto. Agradezco a German Rigau i Claramunt, supervisor del proyecto por parte de la UPV/EHU, la predisposición mostrada y el asesoramiento ofrecido durante el transcurso del mismo. Doy también las gracias a Inigo Etxabe y Beñat Aranburu, supervisores del proyecto por parte de Datik Información Inteligente S.L., por poner a mi disposición todos los medios tecnológicos necesarios para llevar a cabo el proyecto y por el trato ofrecido desde el primer día.

Agradecerles, cómo no, a mis padres José Javier Zabala y María Pilar Barandiaran por el esfuerzo desempeñado para facilitarme, en la medida que les ha sido posible, el camino que he recorrido hasta llegar aquí. Me congratula haber sabido responder satisfactoriamente a la confianza que ellos siempre han depositado en mí. Por todo lo que han hecho y por todo lo que suponen para mí, un beso enorme para los dos.

No quisiera olvidarme de todas las personas que han estado a mi lado durante este maravilloso periplo, a los cuales no me atrevo a mencionar de forma individual por el miedo de dejar a alguno en el tintero. Gracias a los amigos de toda la vida por el apoyo ofrecido durante este camino. Gracias a los compañeros de la facultad, por todo los momentos vividos juntos y en especial a aquellos que durante esta etapa se han ganado a pulso el privilegio a ser parte importante de lo que me resta de existencia.

Por último, pero no por ello menos importante, quisiera evocar a todos los docentes que han tomado parte en mi formación desde aquel Septiembre del 2008 y agradecer a todos ellos el conocimiento compartido y el esfuerzo invertido en mí durante estos años.

Gracias de todo corazón a la gente mencionada en este breve capítulo por haber hecho de mí un mejor profesional y sobre todo una mejor persona.

Resumen

Proyecto Final del Máster en Sistemas Informáticos Avanzados. Estudio de carácter empírico realizado sobre el rendimiento ofrecido por varias tecnologías emergentes en el campo del Big Data en comparación a una base de datos tradicional a la hora de operar en escenarios que requieren un almacenamiento y procesamiento eficaz de volúmenes masivos de datos.

Para llevar a cabo el experimento, dos entornos de prueba totalmente aislados han sido erigidos sobre la misma máquina física. En el primero, se ha configurado un clúster compuesto por cuatro nodos virtuales que operan dentro de una red privada. Dichos nodos han sido dotados de tecnología necesaria para el funcionamiento de Apache Cassandra [**apachecassandra**] y Apache Spark [**apachespark**]. En el segundo, se ha instalado una base de datos MySQL tradicional sobre un único nodo virtual que hereda la potencia total de la máquina física. Una vez habiendo poblado las bases de datos mediante un data-set público de aproximadamente 25GB y diseñado unas consultas acorde a la naturaleza de los datos, se han ejecutado dichas consultas para así cuantificar el tiempo de respuesta que necesitan en cada escenario.

El estudio evidencia que a la hora de trabajar con volúmenes masivos de datos el binomio entre Apache Cassandra y Apache Spark mejora sustancialmente los tiempos de procesado obtenidos con MySQL además de ofrecer una solución totalmente escalable. No obstante, para gozar de las ventajas que ofrecen estas nuevas tecnologías, se antoja necesario un análisis previo de los datos a tratar, aspecto en el que MySQL ofrece una mayor libertad.

Palabras Clave: Big Data, Apache Cassandra, Apache Spark, MySQL, Comparativa.

Índice general

1 Introducción 1
1.1 Contexto 2
1.2 Propuesta 3
1.3 Organización del documento 4

2 Análisis de las tecnologías propuestas 7
2.1 Apache Cassandra 7
2.2 Apache Spark 9
2.2.1 Funcionamiento 9

Índice de figuras

1. Funcionamiento resumido de iPanel 2

2. Ecosistema Spark 9
3. Arquitectura Spark 11
4. Arquitectura Spark 12

Índice de cuadros

Capítulo 1

Introducción

Desde Aristóteles y su libro Segundos Analíticos hasta Galileo, padre de la ciencia moderna, muchos adalides del conocimiento han proclamado que un método de investigación basado en lo empírico y en la medición, sujeto a los principios específicos de las pruebas de razonamiento es el camino para alcanzar la verdad.

Hoy en día, época en la que los avances tecnológico han posibilitado observar y medir de forma exhaustiva un gran abanico de fenómenos, la ingente cantidad de datos que se genera en el proceso es, a veces, intratable por medio de las tecnologías convencionales, y por ende, imposible extraer conocimiento de ellos. El problema, lejos de atenuarse, se acrecienta con el paso del tiempo, ya que, estudios como el realizado por McKinsey Global Institute (MGI) estiman que el volumen de datos que se genera está creciendo un 40 % cada año y auguran que entre 2009 y 2020 se verá multiplicado por 44[1].

Por ello, en los últimos años ha irrumpido la necesidad de encontrar metodologías y herramientas que permitan procesar y extraer el conocimiento que atesora el torrente de información en la cual se encuentra envuelta la sociedad, dando como resultado el nacimiento del Big Data.

El mundo empresarial, por su parte, no se ha mantenido al margen de esta gran revolución. Conscientes de los beneficios que les puede reportar en diferentes aspectos como en el análisis de mercado y calidad de los servicios que ofertan, la gran mayoría de las empresas se han interesado en el Big Data. De un estudio realizado entre los altos ejecutivos de las firmas que lideran el Wall Street se desprende que el 96 % tiene planeadas ciertas iniciativas relacionadas con el Big Data, y el 80 % tiene finalizada alguna[2].

1.1. Contexto

Datik Información Inteligente S.L. es una empresa tecnológica perteneciente al Grupo Irizar que desarrolla soluciones ITS destinadas a la gestión del trasporte, tanto ferroviario como por carretera y movilidad ciudadana.

Uno de los productos estrella de la entidad es el denominado iPanel, concentrador de información que ofrece al operador de transporte servicios de valor añadido en la gestión de la información generada por su flota. El funcionamiento de este servicio se puede resumir mediante la Figura 2:

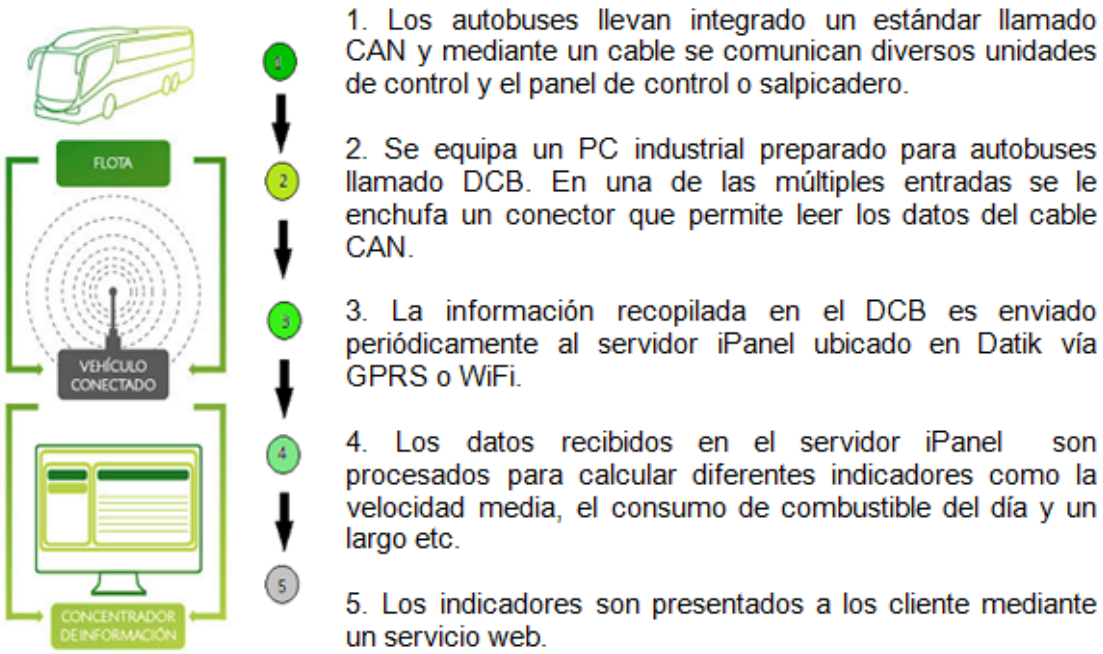

Figura 1: Funcionamiento resumido de iPanel

El incesante aumento en el número de vehículos equipados genera un crecimiento exponencial de los elementos que se han de almacenar y procesar en una base de datos MySQL. Aunque el volumen de datos con el que se trabaja hoy en día no supone riesgo alguno para el funcionamiento del servicio, Datik tiene identificados varios escenarios en los que esto podría cambiar, dando pie al afloramiento de graves problemas.

El primero de todos, es el tiempo necesario para realizar el cálculo de los indicadores. Se trata de un proceso ejecutado una vez al día que atendiendo los datos recopilados en

las últimas 24 hora vuelve a calcular todos los indicadores de iPanel. Ello implica realizar operaciones aritméticas sobre diferentes campos como la velocidad y el consumo de combustible y agrupar los resultados por cliente, flota, vehículo o un espacio temporal. Actualmente, se necesitan varias horas para finalizar la computación, pero, debido al aumento de los datos a tratar, es posible que en un futuro existan graves dificultades para realizar el calculo en menos de 24 horas, invalidando así la funcionalidad de ofrecer los indicadores del último día.

Otro de los problemas, intrínseco al uso de una base de datos centralizada, es el operar sobre un único punto de fallo. Debido a que la mayoría de procesos pasan por dicho punto, existe un alto riesgo de sufrir el denominado efecto dominó, esto es, que la caída provocada por un servicio acarree la del resto. Actualmente, Datik dispone un servidor de réplica capacitado para suplir al primario en caso de que ocurra algo así. No obstante, esta práctica condenar una máquina al ostracismo ya que sus recursos quedan desaprovechados en el 99,95 % del tiempo.

EL último, es la corrupción de datos. Este fenómeno puede suceder (y sucede) debido a un bug, fallo de almacenamiento inesperado, o una caída de MySQL cuando el resultado del checksum de una página es diferente al esperado. Como resultado, podría comprometer seriamente la información que Datik ofrece a sus clientes mediante la aplicación web.

1.2. Propuesta

Para solventar los problemas que Datik prevé, se propone implantar las tecnologías Apache Cassandra y Apache Spark en la empresa y migrar tanto las tablas como los procesos que son parte en el cálculo de los indicadores.

Apache Cassandra es una base de datos distribuida no-sql. Gracias a naturaleza distribuida ayuda a resolver,
Diseñar un plan de migración que defina aspectos tales como:

- Listado de las tablas MySQL que deben ser migradas a Cassandra priorizando las que más rápido crecen y mayor número de consultas intensivas reciben. Por ejemplo, las tablas que contienen la información para el calculo de los indicadores.
- Listado tablas "frontera"
- Ver por cada ejercicio su estado de realización: quiénes lo han terminado, quiénes tienen duda y quiénes no han respondido nada.

- Editar cualquier detalle de un ejercicio en cualquier momento.
- Valorar la realización de un ejercicio a un alumno concreto.

el traspaso de las tablas MySQL que mayor velocidad crecen y mayor número de consultas pesadas reciban a estas nuevas tecnologías. empezando por las que tienen estrecha realción con el calculo de indicadores, ya que, como se ha indicado con anterioridad, es
Por su parte, Apache Cassandra
Por Apache Spark por otra,
Debido a la falta de datos se ha utilizado un dataset publico para emular las condiciones de futuro con las que se va a encontrar datik

1.3. Organización del documento

En esta memoria se ha documentado el desarrollo de la herramienta ***exerClick***, dentro del Trabajo de Fin de Grado (TFG) del autor. En el documento se describe la propuesta, la planificación y gestión que esta lleva consigo, la implementación llevada a cabo y las conclusiones finales.

En este primer capítulo se ha introducido el problema a resolver y se ha explicado la propuesta presentada en este proyecto.

En el capítulo 2 se presenta el Documento de Objetivos de Proyecto (DOP). Este recoge el alcance y las fases y tareas del proyecto, el análisis de riesgos y el análisis de factibilidad.

Una vez en el capítulo 3 se explica la gestión llevada a cabo durante el proyecto. Se presentan las metodologías utilizadas: Metodologías Ágiles e InterMod (adaptada a las necesidades de este proyecto). A continuación se detallan cada una de las iteraciones llevadas a cabo (como parte de la metodología InterMod): duración, objetivos y tareas realizadas. Al final del capítulo se muestra la documentación asociada a las iteraciones y los objetivos, además del seguimiento de tiempo realizado.

A continuación, en el capítulo 4 se detalla el análisis de requisitos. Primero se detallan los requisitos no-funcionales y luego los funcionales (prototipos en papel llevados a cabo durante las primeras iteraciones que dan una visión global del proyecto).

En el capítulo 5 se explica el diseño e implementación llevados a cabo. Se comienza mostrando la estructura de documentos del proyecto, luego el diseño realizado en base al análisis de requisitos del capítulo 4 y finalmente una visión general de la implementación

de la lógica de negocio.

Para finalizar, en el capítulo 6 se presentan las conclusiones, líneas futuras para el proyecto y las lecciones aprendidas.

Fuera de la estructura general de la memoria, tenemos la bibliografia y los apéndices. En estos últimos tenemos las actas de reuniones, las actas de pruebas y la vista de relaciones de la base de datos (de la parte utilizada o creada específicamente para el proyecto).

Capítulo 2

Análisis de las tecnologías propuestas

2.1. Apache Cassandra

a.

Apache Cassandra[12] es una base de datos distribuida desarrollada por DataStax bajo la licencia de Apache que trabaja con datos de tipo clave/valor.

El no tener ni un único punto de fallo le posibilita ofrecer una disponibilidad continua, además de otros servicios como la alta escalabilidad, alto rendimiento, seguridad y una simplicidad operacional que permite reducir el coste total de propiedad. Está siendo utilizada por muchas de las aplicaciones en negocios modernos, siendo la base de datos elegida por un cuarto de las empresas de la Fortune 100[].

Debido a la apuesta realizada a favor de Spark por parte de DataStax, esta empresa ha desarrollado un conector para enlazar Spark y Cassandra utilizando funciones de alto nivel de abstracción.

Será recomendable utilizar Cassandra si:

No será recomendable utilizar Cassandra si:

Como ya se ha apuntado anteriormente, Cassandra en una base de datos NoSQL distribuida que es instalada en todos los nodos del cluster. Aunque físicamente los datos estén repartidos en diferentes máquinas, ofrece la impresión de estar operando como una sola gracias a la capacidad de intercambiar los datos entre los diferentes nodos. Para lograr que todos ellos funcionen de manera coordinada, Cassandra utiliza un protocolo Gossip posibilitando que mediante paso de mensajes periódicamente se dé a conocer el estado de un nodo al resto de los nodos del cluster. Este protocolo es el que posibilita, entre otras cosas, que cada nodo pueda mantener actualizadas sus replicas.

Cassandra, como se acaba de mencionar, posibilita replicar los datos que se almacenan en la base de datos y acceder a ellos mediante un protocolo P2P. Un nodo posee una

réplica para un cierto rango de datos y si un nodo falla, otro que tenga la réplica puede responderá a la petición sin tener que interrumpir el servicio. El número de réplicas que se quieren almacenar se especifica a la hora de crear el keyspace. Otro de los atributos que se debe especificar a la hora de crear un keyspace es la denominada Replica Placemente Startegy, la cual indica la forma en la que se reparten las repicas por el anillo.

homogeneridad de sus nodos

En cuanto a la estructura se refiere, un keyspace es un espacio de nombres para un conjunto de ColumFamily. Por lo general se utiliza uno por aplicación y es considerado el equivalente a una base de datos del modelo relacional. El ColumFamily, a su vez, es capaz de almacenar diferentes columnas, siendo el homologo de una tabla del modelo relacional. Para finalizar, una columna seria una estructura compuesta por clave, valor y timestamp.

Ilustración 5. Estructura de Cassandra

Planificar esta distribución es vital a la hora de operar con los datos en Apache Spark. En el próximo apartado en la cual se hablará del funcionamiento de Apache Spark se volverá a hablar de esto explicando con más detalle las implicaciones que tiene.

Teorema Brewer

Arquitectura homogénea

Cassandra Query Lenguage

Apache Cassandra posee su propio lenguaje de consultas, el denominado CQL, Cassandra Query Lenguage, que se amolda a las características de esta base de datos. La sintaxis de CQL guarda una gran similitud con la de SQL lo cual facilita de forma notoria el salto que supone pasar de trabajar con bases de datos relacionales a distribuidos.

Aún siendo sintácticamente tan parecido a SQL, presenta ciertas restricciones debido a ser un lenguaje de consultas de una base de datos distribuida. Por ejemplo, no ofrece la posibilidad de realizar operaciones como JOIN y es totalmente necesario especificar todas los atributos que componen la clave primaria a la hora de realizar cualquier consulta de filtrado o de actualización en la tabla. La única operación que no cumple esta restricción es un select que contenga la clausula where, ya que, al definir la clave primaria Cassandra indexa de forma automática todos sus componente, posibilitando más tarde hacer uso de ellos en este caso concreto.

Otra de las peculiaridades que presenta CQL es el hecho de ofrecer dos modos distintos de realizar un update. El primero de todos es el mencionado en el párrafo anterior. El segundo posibilita actualizar una columna realizando un insert repitiendo el valor de las claves primarias de una columna ya existente en la base de datos. Esta segunda forma es cómoda a la par de peligrosa porque Cassandra no notifica si una clave primaria ya existe en la base de datos o no, pudiendo un insert desencadenar en un update no deseado.

2.2. Apache Spark

Apache Spark[9] es un proyecto open source de computación en clúster. Desde el principio fue diseñado para poder ejecutar algoritmos iterativos en memoria sin la necesidad de almacenar en disco los resultados intermedios generados durante el proceso. Esta peculiaridad permite que los procesamientos llevados a cabo con Spark puedan llegar a ser, en algunos casos concretos, 100 veces más rápidos que los de MapReduce[10].

A mediados de 2014, coincidiendo con el lanzamiento de la primera versión, alcanzó la cifra de 465 colaboradores, convirtiéndolo en el proyecto más activo entre los relacionados con el Big Data dentro de la Apache Software Fundation.

Apache Spark está compuesto por múltiples y variados componentes que pueden ser utilizados de forma conjunta.

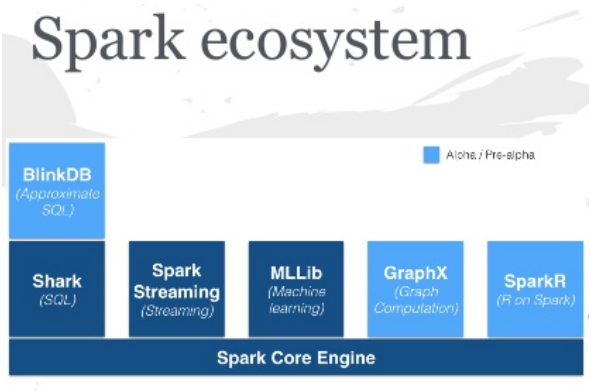

Figura 2: Ecosistema Spark

La base del proyecto es el denominado Spark Core. Proporciona envío distribuido de tareas, planificación y funciones básicas de entrada salida. La abstracción fundamental de programación se llama Resilient Distributed Datasets (RDD)[11], una colección lógica de datos particionados a través de las máquinas que se expone mediante una API integrada en lenguajes como Java, Python y Scala.

2.2.1. Funcionamiento

Para el funcionamiento de Spark, es condición sine qua non que los nodos de la infraestructura tengan acceso a la totalidad de los datos que se desea tratar. Ello implica que para procesar un fichero de 50GB, cada nodo tendría que poseer una copia del mismo

almacenado en su disco. Esta praxis es inviable, ya que más allá de los problemas de consistencia que generaría, para nada es eficiente ocupar la memoria de todos los nodos con información redundante y procesar el fichero entero cuando en realidad se va a hacer uso de una pequeña porción de dichos datos en cada ejecución.

Las bases de datos distribuidas como Cassandra solventan los problemas anteriormente mencionados. Se encargan de distribuir los datos entre diferentes nodos del clúster, ofrecen la posibilidad de acceder a ellos desde cualquier punto y mantienen la consistencia de los mismos a cambio de sufrir una pequeña latencia en el caso de requerir información almacenada en otro nodo de la infraestructura.

Al ejecutar una aplicación que opera con Spark, un componente denominado driver es lanzado. Debido a la necesidad de obtener recursos (CPU y memoria) para llevar a cabo la computación que se le ha encomendado, se comunica con un nodo del clúster que, mediante especificación previa, adopta el rol de maestro. Éste pregunta a todos los nodos que conforman la infraestructura sobre la cantidad de recursos disponibles que poseen y así asignarles los executors correspondiente. Las máquinas que alojen al menos un executor pasan a denominarse worker y a partir de este momento, cada executor podrá comunicarse directamente con el Driver para poder recibir las tareas que éste le envíe.

Un executor es una unidad de trabajo que se encarga de computar las tareas que le encomienda el driver. El número de executors que puede albergar cada worker está directamente relacionado con el número de procesadores que este posee. De la misma forma, es posible repartir la memoria RAM que dispone el nodo worker entre varios executors. Spark permite modificar ambos parámetros programáticamente permitiendo así poder amoldarse a las particularidades de cada ejecución.

Para transferir el código del programa, residente en la máquina del driver, éste adopta el rol de servidor e intenta enviar dicho código a los workers. Si el fichero JAR que contiene el código ha sido recibido correctamente por sus destinatarios, estos responden mediante un ACK y en caso contrario, se vuelve a intentar el envío un número determinado de veces. Una vez llegado al máximo de reintentos, el worker que no haya enviado el ACK es considerado como caído, quedando los executors que albergaba fuera del posterior reparto de tareas.

A la hora de realizar operaciones en Spark, el objeto estrella es el denominado Resilient Distributed Datasets (RDD)[11]. Se trata de una abstracción que mediante diferentes APIs disponibles para Java, Scala y Python permite manipular datos distribuidos por los diferentes nodos del clúster como si estuvieran almacenados de forma local. Este objeto es inmutable, lo cual implica que una vez creado no se le pueden añadir nuevos

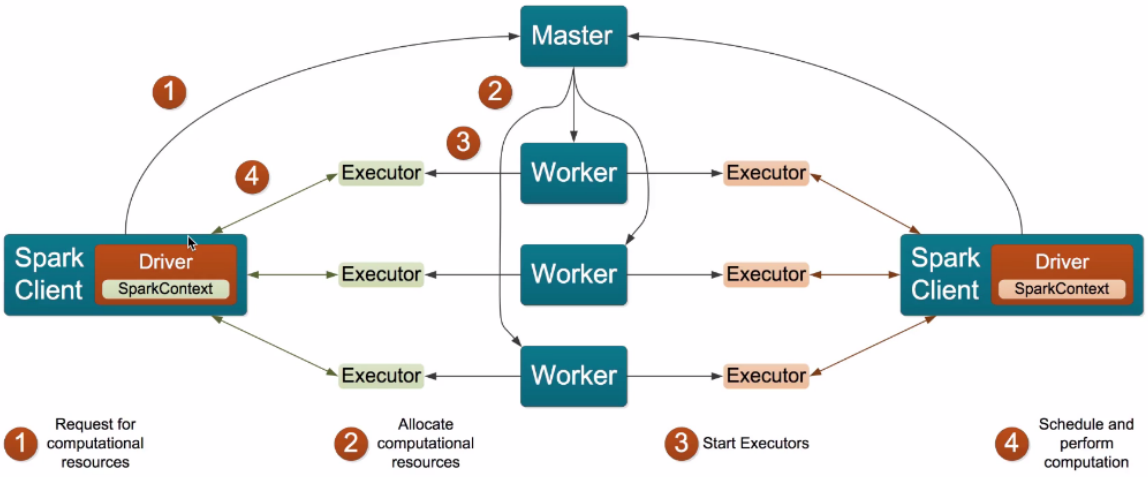

Figura 3: Arquitectura Spark

elementos o eliminar los existentes, solo aplicar transformaciones y acciones sobre el.

Las operaciones que se pueden realizar sobre las RDD se agrupan, tal y como se ha adelantado antes, por transformaciones y acciones. Las primeras transforman un RDD en otro según el criterio indicado y las segundas realizan modificaciones sobre los datos almacenados en dichas RDD. Cabe destacar que las transformaciones en Spark son operaciones "lazy", lo cual implica que en realidad cada nodo memoriza la secuencia de transformaciones que ha de realizar y los procesa cuando una acción es ejecutada.

Una vez terminada la primera fase en la que los executor son creados y enlazados con el driver, éste último empieza a analizar la estructura del código y genera un grafo DAG (Directed Acyclic Graph) con las operaciones que se realizan sobre la RDD. Partiendo de ese grafo genera un job por cada operación de tipo acción que encuentra y dentro de cada job separa la ejecución en diferentes stages según las dependencias que existan entre operaciones. Por último, cada stage es dividido por defecto en unidades de 64MB y a cada unidad resultante se denomina task, el cual es enviado a un executor para ser procesado. El tamaño de cada task puede ser modificado programáticamente, pudiendo de esa forma manipular el número de task que un executor deba ejecutar.

Los datos guardados en las RDD pueden ser almacenados en la memoria para que se puedan procesarse de forma más rápida. Esta opción es conocida como caching. Existen dos formas para cachear los datos: Raw y Serialized. El primero ofrece una mayor rapidez a la hora de procesar los datos, pero consume entre 2 a 4 veces más de memoria. El segundo, en cambio, es más lento en cuanto al procesamiento, pero, la sobrecarga de

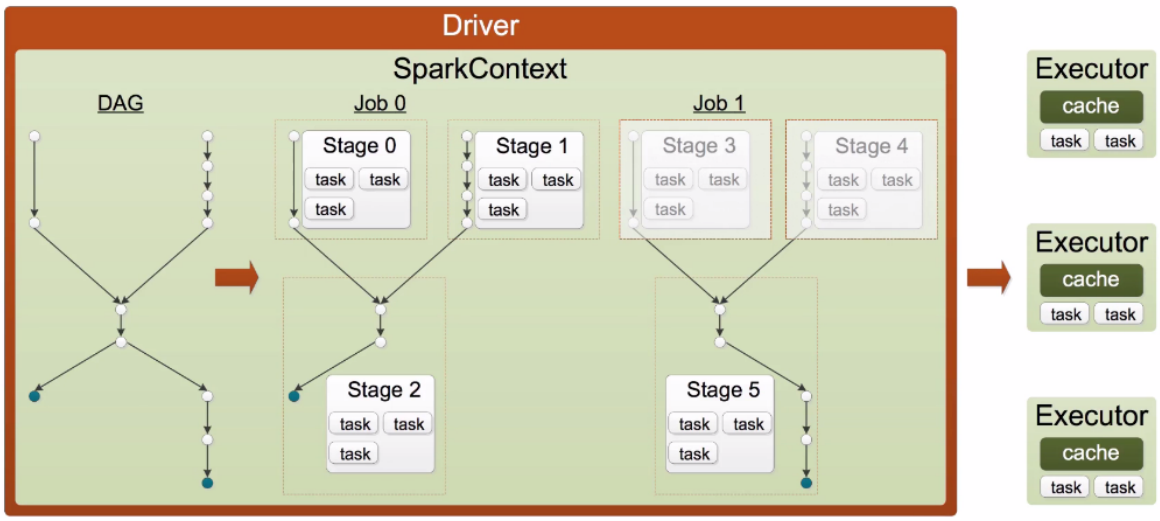

Figura 4: Arquitectura Spark

memoria que supone es casi nula.

Si el tamaño de dato que se quiere cargar en memoria es mayor a la RAM destinada al proceso executor, los datos quedarán en el disco perjudicando gravemente el rendimiento del programa.

El Driver, una vez habiendo recibido los resultados de todas las tareas repartidas, enviará un mensaje a los executors indicando que el procesamiento ha sido finalizado y calculará el resultado final almacenándolo en el propio nodo maestro o en alguna base de datos distribuida como lo es Cassandra en este caso particular.